IQ Data Service

User Guide

IQ Service User Guide

Author: Acconeer

Version 1.0: 2018-07-05

Acconeer AB

Table of Contents

1 Introduction

The IQ Service is one of three services that provide APIs for reading out the radar signal from the Acconeer A111 sensor. The IQ-data can be seen as an extension of the envelope data. In addition to the amplitude, the IQ data also includes information on the phase of the radar signal. The data returned from the IQ Service is represented as complex numbers and the IQ data is typically further processed using various signal processing algorithms. The IQ data can for example be used for measurement of small changes in distance with µm accuracy or for efficient background cancellation.

For applications where the phase information is not needed you may consider using the envelope service or power bin service instead. They both provide amplitude data and are easier to understand and work with compared to the IQ Service. The envelope service provides full resolution, whereas the power bins API provides less processed subsampled amplitude data.

Before using the IQ service, we recommend that you have a basic understanding of complex numbers and how they are used to represent phase and amplitude in signal processing.

2 Setting up the IQ Service

2.1 Initializing the System

The Radar System Services (RSS) must be activated before any other calls are done to the radar sensor service API.

```
if (!acc_rss_activate()) {
    /* Handle error */
}
```

All services in the Acconeer API are created and activated in two distinct steps. In the first creation step the configuration settings are evaluated and all necessary resources are allocated. If there is some error in the configuration or if there are not enough resources in the system to run the service, the creation step will fail. However, when the creation is successful you can be sure that the second activation step will not fail due to any configuration or resource issues. When the service is activated the radar is turned on and the radar data starts to flow from the sensor to the application.

2.2 IQ Service Configuration

Before the IQ service can be created and activated we must prepare a service configuration. First a configuration is created.

```
acc_service_configuration_t iq_configuration;

iq_configuration = acc_service_iq_configuration_create();

if (iq_configuration == NULL) {
    /* Handle error */
}
```

The newly created service configuration contains default settings for all configuration parameters and can be passed directly to the `acc_service_create` function. However, in most scenarios there is a need to change at least some of the configuration parameters.

2.3 Sweep Configuration

The sweep configuration parameters determine the sensor source and how the sweep data will be generated in the sensor. The sweep configuration parameters are common to all services and are therefore handled by a separate sweep configuration. Like other configuration parameters, the sweep parameters have reasonable default values, but in most applications, it is necessary to modify at least some them. To do this we must first obtain a sweep configuration handle.

```
acc_sweep_configuration_t sweep_configuration;

sweep_configuration = acc_service_get_sweep_configuration(iq_configuration);

if (sweep_configuration == NULL) {
    /* Handle error */
}
```

2.3.1 Basic Configurations

Using the sweep configuration handle, we call access functions to set individual configuration parameters such as the sweep start and range.

```
// Set sweep start and length
acc_sweep_configuration_requested_range_set(sweep_configuration, .20, 0.4);
```

When using a connector board with multiple sensors the sensor id must also be set in the sensor configuration.

```
acc_sweep_configuration_sensor_set(sweep_configuration, 1);
```

2.3.2 Set Repetition Mode and Frequency

A repetition mode describes how the sensor behaves when producing data. There are currently two settable modes: Streaming and Max Frequency.

Streaming is the default mode and uses the sensor hardware as source for when to perform sweeps, i.e. the sensor hardware determines the timing. In this mode it is possible to set a desired frequency, see below.

```
// Set repetition mode streaming and the desired sweep frequency
acc_sweep_configuration_repetition_mode_streaming_set(sweep_configuration, 100);
```

The other repetition mode, Max Frequency, allows the host to perform sweeps continuously as fast as possible. The limiting factor on the update rate is now the whole system, i.e. both host and sensor. It is not possible to set a certain frequency in this mode. This mode does not guarantee any form of accurate timing. This mode is not compatible with Power Save Mode A (see chapter on Power Save Mode). Max frequency is set as below.

```
// Set repetition mode max frequency
acc_sweep_configuration_repetition_mode_max_frequency_set(sweep_configuration);
```

2.3.3 Set Sensor Gain (Advanced)

Advanced users can control the receiver gain in the sensor. This may be useful for example in cases when measuring reflections from strong reflectors close to the sensor. The gain value must be between 0.0 and 1.0, where 0.0 is the lowest gain and 1.0 is the highest gain.


```
// Decrease receiver gain
float current_gain;
current_gain = acc_sweep_configuration_receiver_gain_get(sweep_configuration);
acc_sweep_configuration_receiver_gain_set(sweep_configuration, 0.8 * current_gain);
```

2.3.4 Set Power Save Mode (Advanced)

Each power save mode corresponds to how much of the sensor hardware is shutdown between sweeps. Mode A means that the whole sensor is shutdown between sweeps while mode D means that the sensor is in its active state all the time. For each power save mode there will be a limit in the achievable update rate. Mode A will have the lowest update rate limit but also consumes the least amount of power for low update rates.

The update rate limits also depend on integration and range settings so for each scenario it is up to the user to find the best possible compromise between update rate, range and power consumption.

```
acc_sweep_configuration_power_save_mode_set
 (sweep_configuration, ACC_SWEEP_CONFIGURATION_POWER_SAVE_MODE_B);
```

ACC_SWEEP_CONFIGURATION_POWER_SAVE_MODE_A	Maximum power save
ACC_SWEEP_CONFIGURATION_POWER_SAVE_MODE_B	High power save
ACC_SWEEP_CONFIGURATION_POWER_SAVE_MODE_C	Limited power save
ACC_SWEEP_CONFIGURATION_POWER_SAVE_MODE_D	Sensor always active – no power save

3 Capturing IQ Data

3.1 Creating and Activating the Service

After the configuration has been prepared and populated with desired configuration parameters, the actual IQ service instance must be created. During the creation step all configuration parameters are validated and the resources needed by RSS are reserved. This means that if the creation step is successful we can be sure that it is possible to activate the service and get data from the sensor (unless there is some unexpected hardware error).

```
acc_service_handle_t iq_handle = acc_service_create(iq_configuration);

if (iq_handle == NULL) {
    /* Handle error */
}
```

If the service handle returned from `acc_service_create` is equal to NULL, then some setting in the configuration made it impossible for the system to create the service. One common reason is that the requested sweep length is too long, but in general, looking for error messages in the log is the best way to find out why a service creation failed.

When the service has been created it is possible to get the actual number of samples we will get for each sweep. This value can be useful when allocating buffers for storing the IQ data.

```
acc_service_iq_metadata_t iq_metadata;
acc_service_iq_get_metadata(iq_handle, &iq_metadata);
uint16_t data_length = iq_metadata.data_length;
```

It is now also possible to activate the service. This means that the radar sensor starts to do measurements.

```
acc_service_status_t service_status = acc_service_activate(iq_handle);
```

3.2 Reading IQ Data from The Sensor

The easiest way to read the IQ data from the sensor is to call the function `acc_service_iq_get_next`. This function blocks until the next sweep arrives from the sensor and the IQ data is then copied to the `iq_data` array.


```
float complex iq_data[iq_metadata.data_length];
acc_service_iq_result_info_t result_info;

for (int i=0 ; i<50 ; i++)
{
    service_status = acc_service_iq_get_next(iq_handle,
                                             iq_data,
                                             iq_metadata.data_length,
                                             &result_info);


    if (service_status!= ACC_SERVICE_STATUS_OK) {
         /* Handle error */
    }

    /* Process IQ Data */
}
```

3.3 Using Callback Functions to Receive Data (Advanced)

There is an optional way for the application to obtain IQ data from the API. Instead of doing multiple calls to the `acc_service_iq_get_next` function, it is possible to register a callback function. The callback function is then automatically called by Radar System Services whenever there is new IQ data available from the sensor. The main advantage with using a callback function is that the main application is not blocked while waiting for the next sweep. The API overhead also reduced slightly as a copy of the IQ data to a second internal buffer is avoided.

The callback function must be registered in the configuration step before creating the service. A pointer to a user defined data structure is also registered. This pointer will be passed to all calls to the register callback function. The user defined data structure should contain state information that needs to be saved between the calls to the callback function. It is also a good place for storing the length of the IQ data array.

```
typedef struct
{
     uint16_t     data_length;
     uint16_t     some_state_information;
} iq_callback_user_data_t;

iq_callback_user_data_t callback_user_data;

acc_service_iq_iq_float_callback_set(iq_configuration,
                                     &iq_callback,
                                     &iq_callback_user_data);
```

Note that the code in the callback function will execute in a different thread than the rest of the application. This means that access to shared data structures or other shared resources needs to be synchronized properly to avoid race conditions.

```
void iq_callback(const acc_service_handle_t service_handle,
                 const uint16_t *iq_data,
                 const acc_service_iq_result_info_t *result_info,
                 void *user_reference)
{

    iq_callback_user_data_t *callback_user_data = user_reference;

    /* Process IQ Data */


}
```

4 Deactivating and Destroying the IQ service

Call the `acc_service_deactivate` function to stop measurements.

```
service_status = acc_service_deactivate(iq_handle);

if (service_status!= ACC_SERVICE_STATUS_OK) {
    /* Handle error */
}
```

After the service has been deactivated it can be activated again to resume measurements or it can be destroyed to free up the resources associated with the service handle.

```
acc_service_destroy(&iq_handle);
```

Finally, call `acc_rss_deactivate` when the application doesn't need to access the Radar System Services anymore. This releases any remaining resources allocated by RSS.

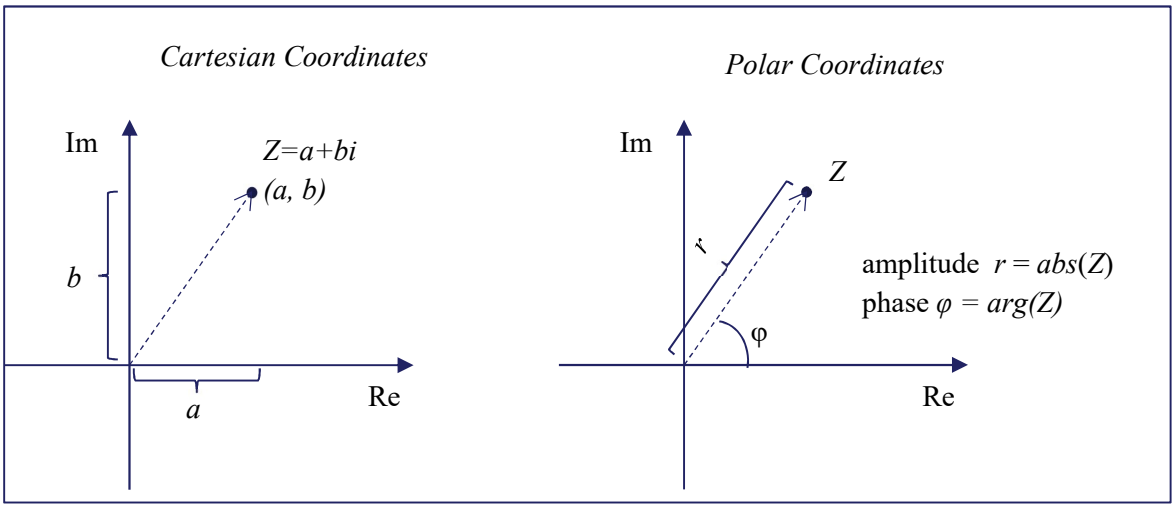
```
acc_rss_deactivate();
```

5 How to Interpret the IQ Data

5.1 Calculating Amplitude and Phase

Each IQ data sample is a complex number consisting of two parts, a real component and an imaginary component. All complex numbers can be written in the form $a + bi$, where a and b are two ordinary real numbers and i is the imaginary unit that can be thought of having the value $\sqrt{-1}$. A complex number $z = a + bi$ is said to have the real part a and the imaginary part b.

Complex numbers can also be seen as points or vectors in the complex plane and be represented in polar coordinates with a radius r and an angle φ. In the context of IQ data, the radius r corresponds to the signal amplitude and φ is the phase of the signal.



The Acconeer IQ data API relay on the c99 representation of complex float. Use the functions `crealf` and `cimagf` to extract the real and imaginary parts of the complex number.

```
#include <complex.h>

float complex z = 2 + 3*I;

float a = crealf(z);
float b = cimagf(z);
```

The functions `cabsf` and `cargf` can be used to extract the amplitude and phase angle φ.

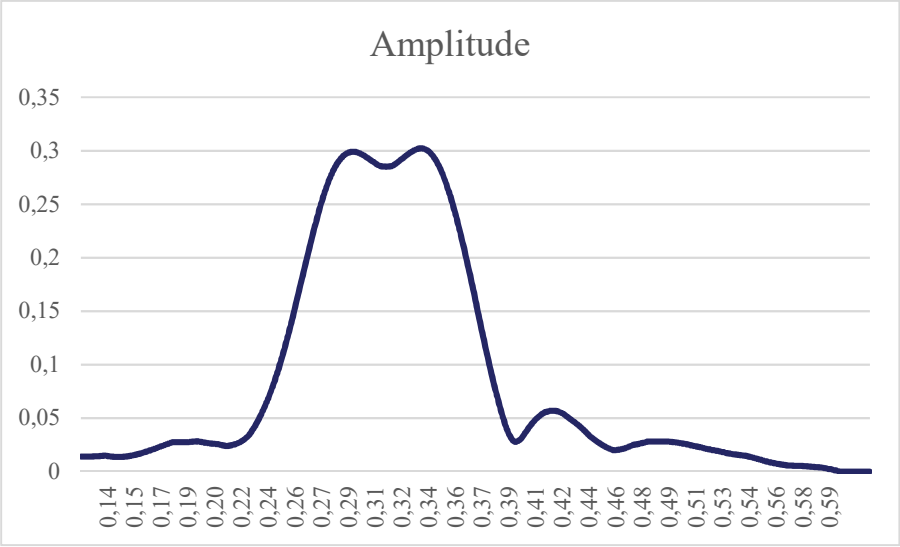
```
float ampliude = cabsf(z); // same as sqrtf(a*a + b*b)
float phase = cargf(z);    // same as atan2f(b, a)
```

The phase difference between two IQ data samples *z1* and *z2* can be calculated using the expression `cargf(z2 * conjf(z1))`.

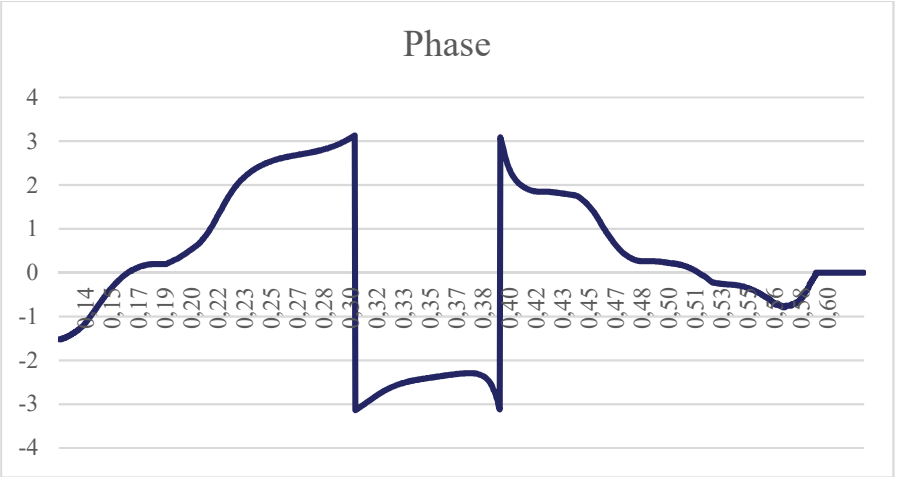
```
float phase_shift = cargf(z2 * conjf(z1));
```


5.2 Plotting amplitude and phase

The graphs below show the amplitude and phase response from an object placed about 28 cm from the sensor. In the amplitude graph we can see shadow reflections 6 and 12 cm behind the object. To achieve as stable phase as possible, we are running the A111 radar sensor in a different mode in the IQ Service compared to the Envelope and Power Bins services. The gets more stable in the IQ service but as a side effect we are getting shadow reflections behind the object.



Note that in the phase plot below, the signal wraps around from π to -π at a distance around 0.30m and then it goes back from -π to π a little bit later.



5.3 IQ metadata

In addition to the array with IQ data samples, the metadata and the result info data structures provide side information that can be useful when interpreting the IQ data.

```
acc_service_iq_metadata_t iq_metadata;
acc_service_iq_get_metadata(iq_handle, &iq_metadata);
```

The most important member variable in the meta data structure is `data_length` which holds the length of the IQ data array. The other member variables, `actual_start_m`, `actual_length_m` and `free_space_absolute_offset`, are needed in applications that needs accurate distance measurements.

metadata struct member	Explanation
`actual_start_m`	Actual start of the sweep - may differ from what requested in the configuration.
`actual_length_m`	Actual length of the sweep - may differ from what requested in the configuration.
`data_length`	Length of the IQ data array.
`free_space_absolute_offset`	Sensor specific offset error. Can be used to reduce the variation in distance measured from different sensors.

6 Micro Motion Measurement Example

In this example we will implement a simple phase tracking algorithm that can detect micro motions about 25 cm from the sensor. It will look at differences in the phase information between consecutive sweeps and from that calculate how much the object has moved.

For each sweep we will look at one sample in the middle of the sweep array so the sweep length can be decreased to a few centimeters. A short sweep range also means that we can run at a high sweep frequency. That is good, because between two sweeps, we can only measure phase differences up to $\pm\pi$ radians – which corresponds to object movements of up to ±1.25mm.

```
/* Set up the IQ service as described in chapter 2 and 3, use the configuration below */

float frequencey = 300;
acc_sweep_configuration_requested_range_set(sweep_configuration, .20, 0.1);
acc_sweep_configuration_repetition_mode_streaming_set(sweep_configuration, frequencey);
```

The phase information is unreliable when the signal amplitude is low, so we wait with the calculations until the amplitude is above a threshold value. This ensures that we are not measuring just noise.

The IQ data sample from the previous sweep is stored in the variable z0 and the sample from the current sweep is stored in z1. The phase difference between the two sweeps is then calculated and we will get the movement between the sweeps by multiplying by `wavelength / (4 * pi)`.

The variable `acc_dist` holds the accumulated distance changes since start of tracking. If the amplitude falls under the threshold the accumulated distance is reset to 0. Note that we are tracking relative movements about 25 cm from the sensor, wo do not measure any absolute distances in this example.

```
const float wavelength = 5.0; // wavelength in mm
const float pi = 3.14159265359;
const float amplitude_threshold = 0.1;
float complex z0 = 0;
float acc_dist = 0;
acc_service_status_t status;

while (true) {
    status = acc_service_iq_get_next(iq_handle, iq_data, iq_metadata.data_length,
                                     &result_info);
    if (status != ACC_SERVICE_STATUS_OK) {
        /* handle error */
    }

    float complex z1 = iq_data[iq_metadata.data_length/2];

    if (cabsf(z1) > amplitude_threshold) {
        if( z0 != 0) {
            float delta_dist = cargf(z1 * conjf(z0)) * wavelength / (4 * pi);
            acc_dist += delta_dist;
            printf("delta distance % 0.2f mm, accumulated distance % 0.2f mm, speed = "
                   "% 0.2f m/s\n", delta_dist, acc_dist, delta_dist * frequencey /1000);
        }
        z0 = z1;
    } else if (z0 != 0) {
        printf("no object detected, resetting tracking\n");
        z0 = 0;
        acc_dist = 0;
    }
}
```

Disclaimer

The information herein is believed to be correct as of the date issued. Acconeer AB ("**Acconeer**") will not be responsible for damages of any nature resulting from the use or reliance upon the information contained herein. Acconeer makes no warranties, expressed or implied, of merchantability or fitness for a particular purpose or course of performance or usage of trade. Therefore, it is the user's responsibility to thoroughly test the product in their particular application to determine its performance, efficacy and safety. Users should obtain the latest relevant information before placing orders.

Unless Acconeer has explicitly designated an individual Acconeer product as meeting the requirement of a particular industry standard, Acconeer is not responsible for any failure to meet such industry standard requirements.

Unless explicitly stated herein this document Acconeer has not performed any regulatory conformity test. It is the user's responsibility to assure that necessary regulatory conditions are met and approvals have been obtained when using the product. Regardless of whether the product has passed any conformity test, this document does not constitute any regulatory approval of the user's product or application using Acconeer's product.

Nothing contained herein is to be considered as permission or a recommendation to infringe any patent or any other intellectual property right. No license, express or implied, to any intellectual property right is granted by Acconeer herein.

Acconeer reserves the right to at any time correct, change, amend, enhance, modify, and improve this document and/or Acconeer products without notice.

This document supersedes and replaces all information supplied prior to the publication hereof.